Social REST API

Specification	Social REST API
Target Application	PLF 4.1
Owner	Thomas Delhoménie
Status	DRAFT - REVIEW- **VALIDATED**

Table of Contents

Please read the common CRUD REST API spec before reading this spec

Resources

The Social REST API provides methods for accessing a resource such as a space or an activity. Here are the descriptions of a full representation of each of them.

User

```
{
  "id": "Id",
```

```
  "href" : "http://{host}:{port}/rest/v1/social/users/{id}",
  "identity": "http://{host}:{port}/rest/v1/social/identities/{identityId}",
  "username": "Username",
  "firstname": "Firstname",
  "lastname": "Lastname",
  "fullname": "Fullname",
  "gender": "Gender",
  "email": "Email",
  "phones": [{
     "phoneType": "Type of phone (home, work, ...)",
     "phoneNumber": "Phone number"
  }],
  "avatar": "href of the avatar file",
  "experiences": [{
     "company": "Company",
     "position": "Position",
     "skills": "Skills",
     "startDate": "Start Date",
     "endDate": "End Date",
     "isCurrent": "Is this experience the current one ?",
     "description": "Description"
  }],
  "position": "Position",
  "ims": [{
     "imType": "Type of IM (skype, GTalk, ...)",
     "imId": "Id of the user for this IM"
  }],
  "urls": ["Array of urls"],
  "deleted": "Is this user deleted ?"
}
```

Users Relationship

```
{
  "id": "id",
  "href" : "http://{host}:{port}/rest/v1/social/usersRelationships/{id}",
  "sender": "http://{host}:{port}/rest/v1/social/users/{userId}",
  "receiver": "http://{host}:{port}/rest/v1/social/users/{userId}",
  "status": "Status of the relationship (pending or confirmed)"
}
```

Space

```
{
  "id": "id",
  "href" : "http://{host}:{port}/rest/v1/social/spaces/{id}",
  "identity": "http://{host}:{port}/rest/v1/social/identities/{identityId}",
  "displayName": "Display Name",
  "url": "Url",
  "groupId": "Group Id",
  "avatarUrl": "Avatar Url",
  "applications": [{
     "id": "Id",
```

```
    "displayName": "Display Name"
  }],
  "visibility": "Visibility (visible or hidden)",
  "subscription": "Subscription (open, validation, close)",
  "managers": "http://{host}:{port}/rest/v1/social/spaces/{id}/users?role=manager",
  "members": "http://{host}:{port}/rest/v1/social/spaces/{id}/users"
}
```

Space Membership

```
{
  "id": "id",
  "href" : "http://{host}:{port}/rest/v1/social/spacesMemberships/{id}",
  "user": "http://{host}:{port}/rest/v1/social/users/{userId}",
  "space": "http://{host}:{port}/rest/v1/social/spaces/{spaceId}",
  "role": "Role of the user in the space (member or manager)",
  "status": "Status of the membership (pending or approved)"
}
```

Activity

```
{
  "id": "Id of the activity",
  "href" : "http://{host}:{port}/rest/v1/social/activities/{id}",
  "identity": "http://host}:{port}/rest/v1/social/identities/{identityId}",
  "title": "Title of the activity",
  "body": "Body of the activity",
  "owner": {
    "identityId": "Identity id of the user who created this activity",
    "href": "http://{host}:{port}/rest/v1/social/users/{userId}"
  },
  "link": "Link attached to the activity",
  "attachments": ["Array of href of the files attached to the activity"],
  "type": "Activity type", // "USER_PROFILE_ACTIVITY", "USER_ACTIVITIES_FOR_RELATIONSHIP",
"USER_COMMENTS_ACTIVITY_FOR_RELATIONSHIP", ... do we have an exhaustive list of activity types ?
  "createDate": "Date of creation",
  "updateDate": "Date of last update",
  "priority": "Priority", // used ?
  "mentions": [{
    "id": "Id of the users mentionned in the comment",
    "href", "http://{host}:{port}/rest/v1/social/users/{userId}"
  }],
  "likes": "http://{host}:{port}/rest/v1/social/activities/{id}/likes",
  "comments": "http://{host}:{port}/rest/v1/social/activities/{activityId}/comments",
  "activityStream": {
    "id": "Id of the identity of the activity stream associated with this activity (id of an user or of a space)",
    "type": "Type of the activity stream associated with this activity ('user' or 'space')"
  }
}
```

Comment

```
{
  "id": "Id",
  "href" : "http://{host}:{port}/rest/v1/social/comments/{commentId}",
  "identity": "http://host}:{port}/rest/v1/social/identities/{identityId}",
  "body": "Body",
  "poster": "Identity of the user who created this comment",
  "createDate": "Date of creation",
  "updateDate": "Date of last update",
  "mentions": [{
     "id": "Id of the users mentionned in the comment",
     "href", "http://{host}:{port}/rest/v1/social/users/{userId}"
   }],
  "activity": "http://{host}:{port}/rest/v1/social/activities/{activityId}"
}
```

Like

```
{
  "href" : "http://{host}:{port}/rest/v1/social/activities/{activityId}/likes/{likerId}",
  "liker" : "Liker Id",
  "likerIdentity": "http://{host}:{port}/rest/v1/social/identities/{identityId}"
}
```

Identity

```
{
  "id": "id",
  "href" : "http://{host}:{port}/rest/v1/social/identities/{id}",
  "remoteId": "Id of the referenced social object (user id, space id, ...)",
  "providerId", "Id of the provider of the identity",
  "globalId": "GlobalId according to the definition of OpenSocial",
  "deleted": "Is deleted ?",
  "profile": ["Array of properties"]
}
```

Relationship

```
{
  "id": "id",
  "href" : "http://{host}:{port}/rest/v1/social/relationships/{id}",
  "sender": "http://{host}:{port}/rest/v1/social/identities/{identityId}",
  "receiver": "http://{host}:{port}/rest/v1/social/identities/{identityId}",
```

```
  "symetric": "Is symetric ?",
  "status": "Status of the relationship (pending, confirmed or ignored)"
}
```

URIs

Base URI : /rest/social

/users					
Verb	**Description**	**Query parameters**	**Input example**	**Output example**	**Notes**

GET	Gets all the users	**q**: query string for filtering. Accepts * wildcard (for example q=jo* retrieves all the users beginnig by 'jo')		<pre>[
				  {
				  "href": "http://localhost:8080/rest/v1/social/users/john",
				  "id": "john",
				  "username": "john",
				  ...
				  }
				]</pre>	

POST	Creates an user		<pre>{
			"username": "mary",
			  ...
			}</pre>
					This action also creates the user on Portal side.
					
					Creates the user if the authenticated user in the /platform/administrators group.

/users/{id}			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets an user with the given id		Returns the user if the request is made by an authenticated user (a social profile is visible by all the users)
PUT	Updates an user with the given id		Updates the user if he is the authenticated user
			This action also deletes the user on Portal side.

DELETE	Deletes an user with the given id		Deletes the user if the authenticated user in the /platform/administrators group

/users/{id}/connections			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the users connected with the user with the given id		Returns the users' connections (list of User resource type) if the request is made by an authenticated user (user's connections are visible by all the users)

/users/{id}/spaces			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the spaces of the user with the given id		Returns the user's spaces (list of Space resource type) in which the user is a member in the following cases: • the given user is the authenticated user • the authenticated user is in the group /platform/administrators (needed for admin operations)

/users/{id}/activities			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the activities of the user with the given id	**type**: **all**, **owner** (activities posted by the user), **connections** (activities posted by user's connections), **spaces** (activities posted in the user's spaces). Defaults to **all**. **after**: filters only activities after this date **before**: filters only activities before this date	Returns an activity in the list in the following cases: • this is an user activity and the owner of the activity is the authenticated user or one of his connections • this is a space activity and the authenticated user is a member of the space Returns a 404 error if there is no authenticated user.
POST	Post an activity in the activity stream of the user with the given id		Post the activity if the given user is the authenticated user

/usersRelationships			
Verb	**Description**	**Query parameters**	**Notes**

GET	Gets all the users relationships	**status**: filters only users' relationships with the given status. Possible values are **all**, **pending**, **confirmed**. Defaults to **all**.
		user: filters only users' relationships of the given user. This filter is only available for administrators (group /platform/administrators). If it is used by a non-administrator user, it is ignored and only the relationships of the authenticated users are returned.
			The following cases can be distinguished :
			• if query param **user** is not defined : return the relationships of the authenticated user
			• if query param **user** is defined...
			  • if the authenticated user is not an administrator : return the relationships of the authenticated user (no matter what the query param **user** is, it is ignored)
			  • if the authenticated user is an administrator : return the relationships of the user defined in the query param **user**
			Returns a 401 error if no user is authenticated.

POST	Add an users relationship		Add the users relationship in the following cases: • the sender or the receiver of the user relationship is the authenticated user • the authenticated user is in the group /platform/administrators (needed for admin operations) Returns a 401 error otherwise.

/usersRelationships/{id}		
Verb	**Description**	**Notes**
GET	Gets an users relationship with the given id	Returns the users relationship in the following cases: • the sender or the receiver of the user relationship is the authenticated user • the authenticated user is in the group /platform/administrators (needed for admin operations) Returns a 401 error otherwise.
PUT	Updates an users relationship with the given id	Updates the users relationship in the following cases: • the sender or the receiver of the user relationship is the authenticated user • the authenticated user is in the group /platform/administrators (needed for admin operations

		) Returns a 401 error otherwise.
DELETE	Deletes an users relationship with the given id	Deletes the users relationship in the following cases: • the sender or the receiver of the user relationship is the authenticated user • the authenticated user is in the group /platform/administrators (needed for admin operations) Returns a 401 error otherwise.

/spaces			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the spaces	**q**: query string for filtering. Accepts * wildcard (for example q=exo* retrieves all the spaces beginnig by 'exo')	Returns a space in the list if the request is made by an authenticated user and in the following cases: • the authenticated user is the super user, as he can see all the spaces • the space is "public" (any user can see it exists) • the authenticated user is member of the space Returns a 401 error if no user is authenticated.
POST	Creates a space		Creates the space if the request is made by an authenticated user. Returns a 401 error otherwise. (any user can create a space currently)

/spaces/{id}		
Verb	**Description**	**Notes**
GET	Gets a space with the given id	Returns the space in the following cases: • the authenticated user is super user • the authenticated user is not a member of the space but the space is "public" (any user can see it exists) • the authenticated user is a member of the space Returns a 401 error otherwise.
PUT	Updates a space with the given id	Updates the space in the following cases: • the authenticated user is super user • the authenticated user is the owner of the space • the authenticated user is a manager of the space Returns a 401 error otherwise.
		Deletes the space in the following cases: • the authenticated user is super user

DELETE	Deletes a space with the given id	• the authenticated user is the owner of the space • the authenticated user is a manager of the space Returns a 401 error otherwise.	

/spaces/{id}/users			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the users of the space with the given id	**role**: **manager**, **member**	Returns the space's users if the authenticated user is a member or manager of the space. Returns a 401 error otherwise.

/spaces/{id}/activities			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the activities of the space with the given id	**after**: filters only activities after this date **before**: filters only activities before this date	Returns the space's activities if the authenticated user is a member of the space. Returns a 401 error otherwise.
POST	Post an activity in the activity stream of the space with the given id		Post the activity only if the authenticated user is a member of the space. Returns a 401 error otherwise.

/spacesMemberships			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the spaces memberships	**status**: filters only spaces' memberships with the given status. Possible values are **all**, **pending**, **approved**. Defaults to **all**. **user**: filters only spaces' memberships of the given user **space**: filters only spaces' memberships of the given space	Returns a spaces membership in the list in the following cases: • the sender of the space membership is the authenticated user • the authenticated user is a manager of the space • the authenticated user is super user (needed for admin operations) Returns a 401 error if no user is authenticated.
POST	Add an user in a space		Add the user in the space in the following cases: • the sender of the space membership is the authenticated user and the space subscription is open • the authenticated user is a manager of the space • the authenticated user is super user (needed for admin operations

			) Returns a 401 error otherwise.

/spacesMemberships/{id}		
Verb	**Description**	**Notes**
GET	Gets a space membership with the given id	Returns the space membership in the following cases: • the user of the space membership is the authenticated user • the authenticated user is a manager of the space • the authenticated user is super user (needed for admin operations) Returns a 401 error otherwise.
PUT	Updates a space membership with the given id	Updates the space membership in the following cases: • the user of the space membership is the authenticated user (restricted by the space subscription policy, for example the user cannot update his own membership to "approved" for a space with a "validation" subscription) • the authenticated user is a manager of the space • the authenticated user is super user (needed for admin operations) Returns a 401 error otherwise.
DELETE	Deletes a space membership with the given id (withdraw an user from a space)	Deletes the space membership in the following cases: • the user of the space membership is the authenticated user • the authenticated user is a manager of the space • the authenticated user is super user (needed for admin operations) Returns a 401 error otherwise.

/identities			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the identities		Returns the identity in the list in the following cases: • the authenticated user has permissions to view the object linked to this identity • the authenticated user is in the group /platform/administrators (needed for admin operations) Returns a 401 error if no user is authenticated.
POST	Creates an identity		Creates the identity if the request is made by an authenticated user in the group /platform/administrators (needed for admin operations) Returns a 403 error if an identity linked to the same remote id already exists, with an explicit message in the body. Returns a 401 error if no user is authenticated.

/identities/{id}		
Verb	**Description**	**Notes**
GET	Gets an identity with the given id	Returns the identity if the authenticated user has permissions to view the object linked to this identity
PUT	Updates an identity with the given id	Updates the identity if the authenticated user has permissions to view the object linked to this identity
DELETE	Deletes an identity with the given id	Deletes the identity if the authenticated user has permissions to view the object linked to this identity

/identities/{id}/relationships			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the relationships of the identity with the given id	**with**: returns the relationship with the given identity	Returns the relationship in the list if the authenticated user can view the object linked to the identity

/relationships			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the relationships	**status**: filters only relationships with the given status. Possible values are **all**, **pending**, **confirmed**, **ignored**. Defaults to **all**.	Returns the relationship in the list in the following cases: • the authenticated user has permissions to view the 2 objects linked to the 2 identities • the authenticated user is in the group /platform/administrators (needed for admin operations) Returns a 401 error if no user is authenticated.
POST	Add a relationship		Creates the relationship in the following cases: • the authenticated user has permissions to view the 2 objects linked to the 2 identities • the authenticated user is in the group /platform/administrators (needed for admin operations) Returns a 403 error if a relationship between the same 2 identities already exists, with an explicit message in the body. Returns a 401 error if no user is authenticated.

/relationships/{id}			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets a relationship with the given id		Returns the relationship if the authenticated user has permissions to view the objects linked to this relationship
PUT	Updates a relationship with the given id		Updates the relationship if the authenticated user has permissions to view the objects linked to this relationship
DELETE	Deletes a relationship with the given id		Deletes the users relationship if the authenticated user has permissions to view the objects linked to this relationship

/activities			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the activities		Returns an activity in the list in the following cases: • the authenticated user is an administrator super user, as he can see all the activities • this is an user activity and the owner of the activity is the authenticated user or one of his connections • this is a space activity and the authenticated user is a member of the space Returns a 401 error if there is no authenticated user.

/activities/{id}		
Verb	**Description**	**Notes**
GET	Gets an activity with the given id	Returns the activity in the following cases: • this is an user activity and the owner of the activity is the authenticated user or one of his connections • this is a space activity and the authenticated user is a member of the space • the authenticated user is super user, since he can see all the activities Returns a 401 error otherwise.
PUT	Updates an activity with the given id	Updates the activity in the following cases: • this is an user activity and the owner of the activity is the authenticated user • the authenticated user is super user, since he can update all the activities Returns a 401 error otherwise.
		Deletes the activity in the following cases:

DELETE	Deletes an activity with the given id	• this is an user activity and the owner of the activity is the authenticated user • the authenticated user is super user, since he can delete all the activities Returns a 401 error otherwise.

/activities/{id}/comments			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the comments of the activity with the given id		Returns a comment in the list if the authenticated user has permissions to see the activity (see GET /activities/{id}) Returns a 401 error if there is no authenticated user.
POST	Post a comment for the activity with the given id		Post the comment only if the authenticated user has permissions to see the activity (see GET /activities/{id}) Returns a 401 error if there is no authenticated user.

/activities/{id}/likes			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets all the likes of the activity with the given id		Returns a like in the list if the authenticated user has permissions to see the activity (see GET /activities/{id}) Returns a 401 error if there is no authenticated user.
POST	Add a like for the activity with the given id		Add the like if the authenticated user has permissions to see the activity (see GET /activities/{id}) Returns a 401 error if there is no authenticated user.

/activities/{id}/likes/{username}			
Verb	**Description**	**Query parameters**	**Notes**
GET	Gets the like of the user with the given username for the activity with the given activity id		Returns the like if the authenticated user has permissions to see the activity (see GET /activities/{id}) Returns a 401 error otherwise.
DELETE	Delete a like from the user with the given username on the activity with the given activity id		Deletes the like in the following cases: • the username is the authenticated user • the authenticated user is super user, since he can delete all the likes Returns a 401 error otherwise.

/comments/{id}		
Verb	**Description**	**Notes**
GET	Gets a comment with the given id	Returns the comment if the authenticated user has permissions to see the related activity (see GET /activities/{id}) Returns a 401 error otherwise.
PUT	Updates a comment with the given id	Updates the comment in the following cases: • the authenticated user is the owner of the comment • the authenticated user is super user, as he can update all the comments Returns a 401 error otherwise. *Is updating a comment allowed at the service level ?*
DELETE	Deletes a comment with the given id	Deletes the comment in the following cases: • the authenticated user is the owner of the comment • the authenticated user is super user, as he can delete all the comments Returns a 401 error otherwise.

Existing REST API

Existing Resource	**Description**	
Activities		
POST {portalName}/social/activities/destroy/{activityId}.{format}	Destroys an activity by its provided Id.	DELETE /rest/v1/s
GET {portalName}/social/activities/{activityId}/likes/show.{format}	Gets a list of users who like the activity.	GET /rest/v1/socia
POST {portalName}/social/activities/{activityId}/likes/update.{format}	Updates the "like" information of the activity.	PUT /rest/v1/socia
POST {portalName}/social/activities/{activityId}/likes/destroy/{identityId}.{format}	Removes the "like" information of the activity.	PUT /rest/v1/socia
GET {portalName}/social/activities/{activityId}/comments/show.{format}	Gets comments on the activity.	GET /rest/v1/socia
GET {portalName}/social/activities/{activityId}/comments.{format}	Gets a limit number of comments on the activity.	GET /rest/v1/socia
GET {portalName}/social/activities/{activityId}.{format}	Gets an activity by its	GET /rest/v1/socia

	Id.	
POST {portalName}/social/activities/{activityId}/comments/update.{format}	Updates the comment information on the activity.	POST /rest/v1/soci /rest/v1/social/comr
GET {portalName}/social/activities/{activityId}/comments/create.{format}	Add comments to an existing activity.	POST /rest/v1/soci
POST {portalName}/social/activities/{activityId}/comments/destroy/{commentId}.{format}	Removes comments on the activity.	DELETE /rest/v1/s
Apps		
GET social/apps/show.{format}	Gets applications from the application registry service of the portal.	*Not covered. Shou*
Identity Services		
GET {portalName}/social/identity/{username}/id/show.json	Gets an identity by a user's name and returns in the JSON format.	GET /rest/v1/socia
Linkshare		
POST social/linkshare/show.{format}	Gets the content of the link by the parameter passed to request.	Not covered
People		
GET social/people/suggest.{format}	Gets users' names that match the input string for suggestion.	GET /rest/v1/socia
	Gets users' information	

GET social/people/getprofile/data.json	that matches the input string.	GET /rest/v1/socia
GET social/people/{portalName}/getConnections.{format}	Gets the information of people who have had connection with the current user.	GET /rest/v1/socia
GET social/people/{portalName}/{currentUserName}/getPeopleInfo/{userId}.{format}	Gets the detailed information of a user on the pop-up, based on his/her username.	GET /rest/v1/socia
GET social/people/getPeopleInfo/{userId}.{format}	Gets a set of information of the target user. The returned information of the user includes full name, position avatar, link to profile and relationship status with the current user who sends request.	GET /rest/v1/socia /rest/v1/social/user
Spaces		
GET {portalName}/social/spaces/mySpaces/show.{format}	Gets the current user's spaces and pending spaces.	GET /rest/v1/socia /rest/v1/social/spac
GET {portalName}/social/spaces/lastVisitedSpace/list.{format}	Provides a way to get the latest spaces ordered by last access and to be able to filter spaces, based on the application Id in the spaces.	*Not covered yet. H*

GET {portalName}/social/spaces/pendingSpaces/show.{format}	Gets a user's pending spaces.	GET /rest/v1/socia
GET {portalName}/social/spaces/suggest.{format}	Suggests the space's name for searching.	GET /rest/v1/socia
Activity Stream		
GET api/social/v1-alpha3/{portalContainerName}/activity_stream/{identityId}.{format}	Gets activities of a defined identity based on an specific activity called baseActivity.	GET /rest/v1/socia /rest/v1/social/spac
GET api/social/v1-alpha3/{portalContainerName}/activity_stream/feed.{format}	Gets the activity stream feed of the authenticated user identity based on an specific activity called "baseActivity".	GET /rest/v1/socia
GET api/social/v1-alpha3/{portalContainerName}/activity_stream/spaces.{format}	Gets space activities of spaces based on an specific activity called "baseActivity".	GET /rest/v1/socia
GET api/social/v1-alpha3/{portalContainerName}/activity_stream/connections.{format}	Gets activities of connections of a specified identity based on an specific activity called "baseActivity".	GET /rest/v1/socia
GET api/social/v1-alpha3/{portalContainerName}/activity_stream/{identityId}ByTimestamp.{format}	Gets activities of a defined identity based on a specific time.	GET /rest/v1/socia /rest/v1/social/spac
GET api/social/v1-alpha3/{portalContainerName}/activity_stream/feedByTimestamp.{format}	Gets the activity stream feed of the authenticated user identity based on a specific time.	GET /rest/v1/socia
GET	Gets space activities	

api/social/v1-alpha3/{portalContainerName}/activity_stream/spacesByTimestamp.{format}	based on a specific time.	GET /rest/v1/socia
GET api/social/v1-alpha3/{portalContainerName}/activity_stream/connectionsByTimestamp.{format}	Gets activities of connections of a specified identity based on a specific time.	GET /rest/v1/social/user
Identity Resources		
GET api/social/v1-alpha3/{portalContainerName}/identity/{identityId}.{format}	Gets the identity and its associated profile by the activity Id.	GET /rest/v1/socia
GET api/social/v1-alpha3/{portalContainerName}/identity/{providerId}/{remoteId}.{format}	Gets the identity and its associated profile by specifying its provider Id and user/space Id. There can be many identities, such as, user identities, and space identities.	*Not covered*
Version		
GET api/social/version/latest.{format}	Gets the latest REST API version of eXo Platform. This version number should be used as the latest and stable one which is considered to include all new features and updates of eXo Platform REST services.	*Not covered*

GET api/social/version/supported.{format}	Gets eXo Platform REST service versions that are supported. This is for backward compatibility. If a client application is using an older eXo Platform REST APIs version, all APIs of the version still can work. The array MUST have the latest to oldest order. For example, [v2, v1, v1-beta3], but not [v1, v2, v1-beta3].	*Not covered*

Examples

List activities of a space

- Gets activities of the space 'myspace'
  curl -X GET http://localhost:8080/rest/v1/social/spaces/myspace/activites

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body: [{"id": "123",
"href" : "http://localhost:8080/rest/v1/social/activities/123 ",
"activityStream": {
"id": "myspace",
"type": "space"
}
,
"...": "..."},
{"id":
"href" : "http://localhost:8080/rest/v1/social/activities/456 ",
"456",
"activityStream": {
"id": "myspace",
"type": "space"
}
,
"...": "..."}]
```

List activities of a user

- Gets activities of the user 'mary'
  curl -X GET http://localhost:8080/rest/v1/social/users/mary/activites

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body: [{"id": "111",
"href" : "http://localhost:8080/rest/v1/social/activities/111 ",
"activityStream": {
"id" : "mary",
"type" : "user"
},
"..." : "..."},
{"id" : "222",
"href" : "http://localhost:8080/rest/v1/social/activities/222 ",
"activityStream": {
"id" : "mary",
"type" : "user"
},
"...": "..."}]
```

Post an activity in a space

- John post an activity in the space myspace
  curl -X POST http://localhost:8080/rest/v1/social/spaces/myspace/activities -d '
  {
  "title": "Breaking News !",
  "body": "New eXo Platform Release !!!",
  "...": "..."
  }'

```
HTTP/1.1 201 Created
Location: http://localhost:8080/rest/v1/social/activities/465456
Content-Type: application/json;charset=UTF-8
```

Connection between 2 users

- Mary sends a connection request to John
  curl -X POST http://localhost:8080/rest/v1/social/usersRelationships -d '
  {
  "sender": "http://localhost:8080/rest/v1/social/users/mary ",
  "receiver": "http://localhost:8080/rest/v1/social/users/john ",
  "status": "pending"
  }'

```
HTTP/1.1 201 Created
Location: http://localhost:8080/rest/v1/social/usersRelationships/123465
Content-Type: application/json;charset=UTF-8
```

- John accepts the connection request
  curl -X PUT http://localhost:8080/rest/v1/social/usersRelationships/123465 -d '

```
{
  "sender": "http://localhost:8080/rest/v1/social/users/mary ",
  "receiver": "http://localhost:8080/rest/v1/social/users/john ",
  "status": "confirmed"
}'
```

```
HTTP/1.1 200 OK
```

Add an user in a space

- John joins the space HR
  curl -X POST http://localhost:8080/rest/v1/social/spacesMemberships -d '
  {
  "user": "http://localhost:8080/rest/v1/social/users/john ",
  "space": "http://localhost:8080/rest/v1/social/spaces/exo_hr ",
  "role": "member",
  "status": "approved"
  }'

```
HTTP/1.1 201 Created
Location: http://localhost:8080/rest/v1/social/spacesMemberships/987564
Content-Type: application/json;charset=UTF-8
```

Remove an user from a space

curl -X DELETE http://localhost:8080/rest/v1/social/spacesMemberships/987564

```
HTTP/1.1 200 OK
```

Remove a relationship between 2 generic identities (from their identity ids)

- Gets the relationship between identity '1' and identity '2'
  curl -X GET http://localhost:8080/rest/v1/social/identities/1/relationships?with=2

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body: {"id": "id",
"href" : "http://localhost:8080/rest/v1/social/relationships/123 ",
"...": "..."}
```

- Remove the relationship
  curl -X DELETE http://localhost:8080/rest/v1/social/relationships/123

```
HTTP/1.1 200 OK
```

Remove an user from a space (in the case we don't have the usersRelationship and spacesRelationship resource types)

- Gets the user 'john' (in order to retrieve his identity id)
  curl -X GET http://localhost:8080/rest/v1/social/users/john

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body: {"id": "Id",
"href" : "http://localhost:8080/rest/v1/social/users/john ",
"identity": "http://localhost:8080/rest/v1/social/identities/1 ",
"username": "john",
"...":"..."}
```

- Gets the space 'marketing' (in order to retrieve its identity id)
  curl -X GET http://localhost:8080/rest/v1/social/spaces/marketing

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body: {"id": "Id",
"href" : "http://localhost:8080/rest/v1/social/spaces/marketing ",
"identity": "http://localhost:8080/rest/v1/social/identities/2 ",
"displayName": "Marketing",
"...":"..."}
```

Then refer to the use case "Remove a relationship between 2 generic identities (from their identity ids)"

Create a business social object (sales opportunity) and connect an user to it

- Create the opportunity
  curl -X POST http://localhost:8080/rest/v1/social/identities -d
  {
  "remoteId": "myOpportunity",
  "providerId", "myFavoriteCRM",
  "globalId": "??",
  "deleted": "false",
  "profile": [
  "customer": "Acme",
  "createdDate": "",
  "status": "..."
  ]
  }'

```
HTTP/1.1 201 Created
Location: http://localhost:8080/rest/v1/social/identities/1
Content-Type: application/json;charset=UTF-8
```

- Connect the user to the opportunity (gets the identity id of the user, then creates a relationship between the 2 identities)
  curl -X GET http://localhost:8080/rest/v1/social/users/john

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Body: {"id": "Id",
"href" : "http://localhost:8080/rest/v1/social/users/john ",
"identity": "http://localhost:8080/rest/v1/social/identities/2 ",
"username": "john",
"...":"..."}
```

curl -X POST http://localhost:8080/rest/v1/social/relationships -d '
{
"sender": "http://localhost:8080/rest/v1/social/identities/1 ",
"receiver": "http://localhost:8080/rest/v1/social/identities/2 ",
"symetric": "false",
"status": "confirmed"
}'

```
HTTP/1.1 201 Created
Location: http://localhost:8080/rest/v1/social/relationships/14
Content-Type: application/json;charset=UTF-8
```

Like an activity

- John likes the activity 1234
  curl -X POST http://localhost:8080/rest/v1/social/activities/1234/likes -d '
  {
  "liker": "john"
  }'

```
HTTP/1.1 201 Created
Location: http://localhost:8080/rest/v1/social/likes/465456
Content-Type: application/json;charset=UTF-8
```

Questions

- does it make sense to allow POST on /activities ? (as we already have POST on /spaces/{id}/activities and /users/{id}/activities)